On the Importance of Container Image Placement for Service Provisioning in the Edge

Jad Darrous, Thomas Lambert, Shadi Ibrahim

On the Importance of Container Image Placement for Service Provisioning in the Edge

Jad Darrous
Inria, ENS de Lyon, LIP
Lyon, France
jad.darrous@inria.fr

Thomas Lambert
Inria, IMT Atlantique, LS2N
Nantes, France
thomas.lambert@inria.fr

Shadi Ibrahim
Inria, IMT Atlantique, LS2N
Nantes, France
shadi.ibrahim@inria.fr

***Abstract*—Edge computing promises to extend Clouds by moving computation close to data sources to facilitate short-running and low-latency applications and services. Providing fast and predictable service provisioning time presets a new and mounting challenge, as the scale of Edge-servers grows and the heterogeneity of networks between them increases. This paper is driven by a simple question: can we place container images across Edge-servers in such a way that an image can be retrieved to any Edge-server fast and in a predictable time. To this end, we present KCBP and KCBP-WC, two container image placement algorithms which aim to reduce the maximum retrieval time of container images. KCBP and KCBP-WC are based on k-Center optimization. However, KCBP-WC tries to avoid placing large layers of a container image on the same Edge-server. Evaluations using trace-driven simulations show that KCBP and KCBP-WC can be applied to various network configurations and reduce the maximum retrieval time of container images by 1.1x to 4x compared to state-of-the-art placements (i.e., Best-Fit and Random).**

***Index Terms*—Edge Computing, Services provisioning, Container Image, Placement.**

I. Introduction

Cloud computing has played a significant role in increasing the agility of web service deployment while minimizing the cost by leveraging the economy of scale. However, these centralized Cloud data-centers might not be a suitable deployment platform for every type of application. For example, latency sensitive applications should run physically close to the source of data (e.g., smart city applications) and data analysis on distributed data has to be done in place without moving the data to a central place for performance and privacy concern (i.e., video stream analysis). To overcome these limitations and provide new opportunities, a more distributed Cloud model has been emerging, named Edge computing [1]. Edge computing, promising to extend Clouds by moving computation close to data sources, has been successfully deployed and utilized in practice to facilitate short-running and low-latency applications and services. For instance, running smart city applications at the Edge shows to be up to 56% more efficient compared to Clouds [2].

In general, services are deployed as containers (a lightweight virtualization technology) in the Edge, therefore, *container images* are needed to run those services. However, unlike Clouds, where all *container images* are available locally, provisioning (deploying) a service in the Edge usually requires pulling the corresponding container image over the wide area network (WAN) from a central repository. This may result in long provisioning time; depending on the container images sizes and the network bandwidth [3]. Usually, the bandwidth between Cloud and Edge-servers can be a few tens of Mb/s [4], hence, an image of a size of 1 GB needs at least 100 seconds to be transferred to the Edge-server if the bandwidth is 10 MB/s. This is unacceptable for most Edge services and applications, especially for short-running services and latency-sensitive applications. For example, we often scale-out live video stream analytics to handle data burst; thus, as the response time of those applications is in order of milliseconds, it is not acceptable to wait for hundreds of seconds to provision a new container.

As Edge-servers have limited storage capacity, storing all the images locally is not feasible, especially as the local storage should be exploited to store application data instead of container images. However, given that Edge-servers are featured with high network bandwidth among them compared to the bandwidth with Clouds (e.g., running distributed IoT application on a cluster of Edge-servers is 5.3x times faster than running it on Clouds [5]); a possible solution is to distribute container images across Edge-servers, thus, the network bandwidth and available storage (a small fraction) can be exploited efficiently. Unlike most approaches in the Cloud, container image retrieval in Edge environments needs to be aware of the network heterogeneity between Edge-servers. Even worse, container images and layers are highly heterogeneous. As a result, the retrieval time will depend on the image sizes and the distribution of their layers (and the replicas: usually layers are replicated for performance and fault-tolerance); hence, it is hard to predict the retrieval time of a container image. In this paper, we argue that the initial placement is important for fast service provisioning in Edge environments. Moreover, a service can be provisioned on multiple Edge-servers and an Edge-server may host multiple applications (e.g., camera devices host multiple applications [6]), therefore, it is essential to ensure predictable provisioning time as well. Our work tackles this problem (i.e., fast and predictable service provisioning in the Edge) by introducing novel placement algorithms that target reducing the maximum retrieval time of an image to any Edge-server. *To the best of our knowledge, no previous studies have worked on container image placement in*

Edge environments or targeted reducing the maximum retrieval time of container images.

Contributions. In an attempt to demonstrate the importance of container image placement across Edge-servers, in this paper, we propose and evaluate through simulation two novel container image placement algorithms based on k-Center optimization. In particular, we introduce a formal model to tackle down the problem of reducing the maximum retrieval time of container images, which we denote as MaxImageRetrievalTime. Based on the model, we propose KCBP and KCBP-WC, two placement algorithms which target reducing the maximum retrieval time of container images to any Edge-server. While KCBP is based on a k-Center solver (i.e., placing k facilities on a set of nodes to minimize the distance from any node to the closet facility) which is applied on each layer and its replicas (taking into account the storage capacities of the nodes), KCBP-WC uses the same principle but it tries to avoid simultaneous downloads from the same node. More precisely, if two layers are part of the same image, then they cannot be placed on the same nodes. We have implemented our proposed algorithms alongside two other state-of-the-art placement algorithms (i.e., Best-Fit and Random) in a simulator written in Python. We simulate the behavior of the algorithms on realistic and synthetics networks with a dataset of container images from IBM production Cloud data-center [7]. Simulation results show that the proposed algorithms can outperform state-of-the-art algorithms by a factor of 1.1x to 4x depending on the characteristics of the networks. For example, on the Sanet network [8], we reduce the retrieval time by 13% and 18% compared to Best-Fit and Random, respectively.

The remainder of this paper is organized as follows. First, Section II introduces Edge computing and container image management, followed by a state-of-the-art in Section III. Next, the problem alongside the proposed algorithms are formalized in Section IV. Simulation methodology is discussed in Section V. Then, the obtained results are presented in Section VI and discussed in more details in Section VII. Finally, Section VIII concludes this study.

II. Background

A. Edge computing

The Edge is an extension to the Cloud that includes all the devices that are virtualized and placed on/near the data sources ranging from surveillance cameras [6], to Cloudlet [9] and micro data-center [10]. As in traditional Cloud, those resources are leveraged by the users to run their services. Unlike Clouds, Edge-servers have limited computation and storage capacity. However, they are featured with high network bandwidth among them compared to the bandwidth to the Clouds. Hence, they can act as one cluster and serve a wide range of smart city applications [11]: despite the data exchange across Edge-servers, by eliminating the data transmission to Clouds, the response time of face recognition application can be reduced from 900ms to 169ms [5].

B. Container Image management

Most services run in virtualized environments in both Clouds and Edge. While Virtual Machines (VMs) are widely used in Clouds; containers, being more lightweight in terms of provisioning times and image sizes, are often deployed in the Edge [12], [13]. Many container technologies have been developed recently such as LXC [14] and Docker [15]. Container images are structured as layers. Each image consists of an ordered set of layers that are stacked one on top of the other, and a *manifest* that maintains the layers of that image. Each layer is a collection of files that are stored as a gzip-compressed tar file. Layers could range in size from few Kilobytes to several Megabytes [7]. The *registry* is the component responsible for container image management in the Docker ecosystem. It stores all the layers (from all the container images). Since the same layer could be shared by multiple container images, Docker registry uses layers level deduplication to reduce the size of stored data. Docker registry could be a remote online registry (e.g., Docker Hub [16]), a central private registry, or a local registry on the same machine of the Docker daemon.

III. Related Work

Container image management. Slacker [17] uses chunks level deduplication to speed up container provisioning by transferring the accessed blocks on demands. This is motivated by the fact that just a small fraction of the image is needed to boot the container. However, dividing Docker images into chunks may break the layer structure among the layers and could bring integrity issues if these chunks are transferred over an insecure network. BitTorrent protocol has been used to distribute container layers in a single data-center setup [18]. It has been shown that a great reduction in provisioning time could be achieved when the same image is requested by a large number of machines simultaneously. However, for single image provisioning, no improvement can be achieved. Moreover, BitTorrent protocol does not guarantee a predictable performance.

Docker image placement and container provisioning have been studied in [19]. The authors in [19] propose a collaborative docker registry where the private registries on the cluster (compute) nodes can collaborate to store and retrieve docker images instead of relying on a central or remote registry. Also, the authors opt for layer placement rather than complete image placement to avoid redundancy in common layers. For layer placement, they employ simple heuristics by sorting the layers and the nodes Ascending/Descending. Bipartite graphs have been used to balance the retrieval of missing layers from multiple nodes. All the layers are 3-way replicated. While this work targets homogeneous and single cluster environments, this paper deals with heterogeneous network environments.

In contrast to the aforementioned works, which focus on single data-center setup, in this study, we opt for fast and predictable image provisioning in Edge environments, where the network links between the Edge-servers are heterogeneous.

Virtual Machine Image (VMI) management. Most of the work on VMI management focuses on reducing the storage capacity required to store VMIs by leveraging deduplication [20], [21] and improving the provisioning time of VMs by exploiting the already available chunks of the host machine [22], [23], [24]. Few studies have targeted geo-distributed VMI retrieval [25], [26] where the goal is to minimize the transfer time of VMIs over a heterogeneous WAN. *While all the aforementioned works target VMI management and focus on data retrieval, in this work, we investigate the importance of container image placement on the provisioning time.*

IV. Container Image Placement

In this section, we introduce the formal model we use to study the container image placement problem. We also introduce the two heuristics we propose to distribute a set of replicated layers through a network (across Edge-servers). In Section IV-A, we focus only on a set of individual layers and try to optimize the maximal retrieval time. In Section IV-B, we show how to extend the problem to a set of images (that are themselves sets of layers) and how to adapt our placement.

A. Layer Placement

1) Formal Model: For the moment, we focus on layers placement and put aside complete images. First, a *layer* l_i is defined by its *size* s_i (i.e., its storage cost) and its *replication number* n_i (i.e., how many times a layer is replicated). We denote by $\mathcal{L}_i = l_i^1, \ldots, l_i^{n_i}$ the replicas of l_i. In the following, the complete set of layers is denoted as $\mathcal{L}$ and $\mathcal{L}^R$ represents the set of replicas ($\mathcal{L}^R = \bigcup \mathcal{L}_i$).

In our model, the network is defined as a set of *nodes* V that are fully connected (thus a complete graph). We denote as c the *storage capacity* of all nodes (the allocated storage space on each node). For all the pairs of nodes u, v, we denote as b_{uv} the bandwidth between these two nodes (if $u = v$ then $b_{u,v} = +\infty$).

Given a set of layers $\mathcal{L}$, a set of nodes V and a storage capacity c, we define a *placement* as a function σ from $\mathcal{L}^R$ to V (we want to place all replicas). A placement is said to be *valid* if for each $u \in V$, $\sum_{l_i^k \in \sigma^{-1}(u)} s_i \leq c$ (i.e., the sum of the sizes of stored replicas does not exceed the storage capacity) and for each $k, k' \in [1, n_i]$, $\sigma(l_i^k) \neq \sigma(l_i^{k'})$ (i.e., replicas of the same layer have to be placed on different nodes).

From a valid placement, we derive the *retrieval time* of a layer l_i on a node u as follows. Let u_i^σ be the node owning a replica of l_i that is the closest to u (i.e., with maximal bandwidth, formally $u_i^\sigma = \arg(\max_{v \in \sigma(\mathcal{L}_i)} b_{uv})$. The retrieval time of l_i on u is thus $T_i^u = \frac{s_i}{b_{uu_i^\sigma}}$. Our goal here is to minimize the maximal retrieval time for all layers on all nodes. We denote this problem MaxLayerRetrievalTime.

***Problem** 1 (MaxLayerRetrievalTime):* Let V be a set of nodes with storage capacity c and $\mathcal{L}$ be a set of layers. Return a valid placement that minimizes: $\max_{u \in V,\ l_i \in \mathcal{L}} T_i^u$.

MaxLayerRetrievalTime is close to the k-Center problem, that aims to place facilities on a set of nodes to minimize the distance from any node to the closest facility. See below for a formal definition.

***Problem** 2 (k-Center):* Given a set V with a distance function d (defined between all elements of V), and a parameter k, return a set $S \subseteq V$ such that $|S| = k$ that minimizes: $\max_{u \in V} \min_{v \in S} d(u, v)$.

MaxLayerRetrievalTime is similar to the k-Center problem if considering only one layer. k-Center problem is known for being NP-complete [27] and thus MaxLayerRetrievalTime is also NP-complete. In addition, it has been proven that the best possible approximation is a 2-approximation (unless P=NP) [28].

The solution we introduce to solve MaxLayerRetrievalTime is based on a solver for k-Center. The basic principle of this heuristic, KCBP (k-Center-Based Placement), is to sort the layers in descending order by their sizes and then use this solver several times to place replicas, one layer after another. The distance used is the inverse of bandwidths. A pseudo-code of KCBP is provided in Algorithm 1.

For our implementation, we use Scr, a polynomial k-Center problem solver that was introduced by Robič and Mihelič [29]. Scr is based on a pruning technique (i.e., removing some edges and find a solution on the induced subgraph) and a Dominant Set problem solver (as the Dominant Set problem is also NP-complete, Scr relies on a heuristic). Note that Scr is a 2-approximation, but its experimental average approximation factor is far better and as far as we know the best among polynomial heuristics: 1.058 for Scr on a classical benchmark for graph partitioning [29].

The time complexity of Scr is $O(m^2 \log m)$, where m is the number of nodes. Hence, the overall complexity of KCBP is $O(|\mathcal{L}||V|^2 \log |V|)$.

Algorithm 1: KCBP $(\mathcal{L}, V, c)$

```
Sort L by decreasing size ;
foreach u ∈ V do
    c_u = c ;
foreach l_i ∈ L do
    V' = {u ∈ V, c_u ≥ s_i} ;
    S = Scr(V', n_i) ;
    k = 1 ;
    foreach u ∈ S do
        σ(l_i^k) = u ;
        k++ ;
        c_u ← c_u − s_i ;
return σ
```

B. Image Placement

MaxLayerRetrievalTime focuses on layers. However, we target the retrieval of complete container images.

1) Formal definition: We define an *image* as a set of layers $I_j = \{l_{i_1}, \ldots, l_{i_q}\}$. The complete set of images is denoted as $\mathcal{I}$. To retrieve an image I_j, a node u has to download a

replica of each layer that is in I_j. As a first approximation, we could consider the downloads are done in parallel and thus the retrieval time is defined by the largest retrieval time among this different layers (as in MaxLayerRetrievalTime). However, multiple downloads from a same source may degrade the performance by reducing the bandwidth. Therefore, in our model, we consider that if a node requests an image that requires several layers where the closest replicas are on the same node, then the download of these replicas is made sequentially (that is equivalent to do it in parallel with shared bandwidth). More formally, given an image I_j, a valid placement σ, and a node $u \in V$, let $V^{\sigma}_{u,I_j} = \{v \in V,\ l_i \in I_j \text{ and } u^{\sigma}_i = v\}$ the set of nodes that are the closest nodes to u for at least one replica of the layers of I_j. The retrieval time of an image I_j is thus:

$$T^u_{I_j} = \max_{v \in V^{\sigma}_{u,I_j}} \frac{\sum_{i,u^{\sigma}_i = v} s_i}{b_{uv}}.$$

We define now MaxImageRetrievalTime where the goal is to minimize the maximal retrieval time of a set of images.

***Problem** 3 (MaxImageRetrievalTime):* Let V be a set of nodes with storage capacity c and $\mathcal{I}$ be a set of images. Return a valid placement that minimizes: $\max_{u \in V, I_j \in \mathcal{I}} T^u_{I_j}$.

2) Without-Conflict: If two layers are part of the same image, then their replicas should not be on the same nodes. However, applying this constraint to all layers can lead to a huge spreading of the replicas and even to a lack of eligible nodes (i.e., nodes with enough remaining storage capacities and have no conflicting layers). Thus, we limit the number of layers that are concerned. More precisely, we add a parameter f that is a percentage of the layers. If a layer l_i is among the $f\%$ largest layers, then this layer cannot be placed on a node that already has a replica of a layer $l_{i'}$ which belongs to the same image I_j. We denote this algorithm KCBP-WC (KCBP-Without-Conflict) and a pseudo-code is provided in Algorithm 2.

C. Limitations

To simplify the scenario, we assume that there is a direct link between each pair of Edge-servers. Edge networks are more complex with more network components (switches, routers, etc) with complex and sometimes redundant paths between nodes. Software level approached as Traffic Engineering [30] and Software-Defined Networking (SDN) [31] are widely employed to abstract the physical network topology from the application and ensure some network properties (e.g., bandwidth) between nodes. Hence, the general trend of the relative performance between the placement algorithm could be preserved. Note that the main goal of this work is to shed light on the importance of container image placement in the Edge. Moreover, we plan to address the problems of path sharing and the load variation of the network during the retrieval time (by introducing a network-aware layer retrieval approach, similar to [26]).

Algorithm 2: KCBP-WC $(\mathcal{I}, \mathcal{L}, V, c, f)$

```
Sort L by decreasing size ;
foreach u ∈ V do
    c_u = c ;
foreach l_i ∈ L do
    V' = {u ∈ V, c_u ≥ s_i} ;
    if l_i is one of the f% largest layer then
        foreach I_j ∈ I such that l_i ∈ I_j do
            V' ← V' \ {u ∈ V', ∃ l_i' ∈ I_j, σ(l_i'^k) = u}
    S = Scr(V', n_i) ;
    k = 1 ;
    foreach u ∈ S do
        σ(l_i^k) = u ;
        k++ ;
        c_u ← c_u − s_i ;
return σ
```

V. Simulation Methodology

We developed a simulator in Python to evaluate the performance of the two proposed placement algorithms on different networks and using real container images dataset.

Our simulator is written in Python and the source code is publicly available at https://gitlab.inria.fr/jdarrous/image-placement-edge.

A. Network topology

We generate synthetic networks with different bandwidth characteristics and use real-world networks topologies for our evaluation. All the networks are described in Table I.

Synthetic networks. As we consider that network topologies have no interference, we generate complete graphs (i.e., there is a direct link between each pair of nodes) and then assign bandwidths to these links. Four distributions have been considered: (1) *Homogeneous*: where all the links have the same bandwidth. (2) *Low*: where the majority of the links have low bandwidth. (3) *High*: where the majority of the links have high bandwidth. (4) *Uniform*: where the links bandwidths follow a uniform distribution between 8Mbps and 8Gbps.

Real-world networks. In addition to synthetic networks, we choose two real-world networks to demonstrate the applicability of our algorithms. We select the national networks of France (Renater) and Slovakia (Sanet) [8]. To compute the bandwidth between two nodes that are not directly connected, we do the following: suppose that n is the minimum number of nodes to reach one node from the other, and $minb$ is equal to minimum bandwidth of the links that form the shortest unweighted path between these two nodes. Accordingly, the bandwidth between these two nodes is set to $minb * 0.95^n$.

B. Container images dataset

Container images and their corresponding layers are retrieved from publicly released IBM Cloud traces [7]. We extract the images and layers from the traces of *frankfort*

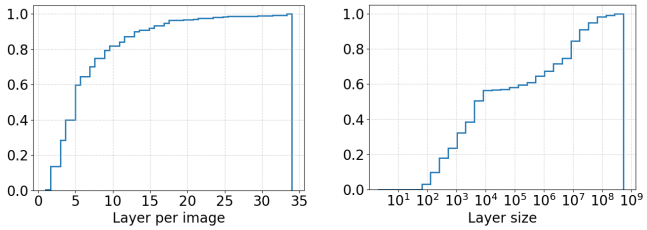
TABLE I
THE SIZES AND LINKS BANDWIDTHS CHARACTERISTICS OF THE STUDIED NETWORKS.

Network	Number of nodes	Links bandwidths (bps)				
		min	25th	median	75th	max
Homogeneous	50	4G	4G	4G	4G	4G
Low	50	8M	763M	1G	2G	8G
High	50	478M	5G	6G	7G	8G
Uniform	50	8M	2G	4G	6G	8G
Renater	38	102M	126M	132M	139M	155M
Sanet	35	63M	6G	8G	8G	10G

data-center. The data set is composed of 996 images with 5672 layers, see Table II. The majority of images (56%) have between 5 and 15 layers, however, some images are composed by up to 34 layers, see Fig. 1a. The layers are highly heterogeneous in size (vary from $100B$ to $1GB$). Moreover, 30% of the layers are larger than $1MB$, see Fig. 1b.

TABLE II
THE CHARACTERISTICS OF THE CONSIDERED IMAGES DATASET.

Total #images	996
Total size of images	93.76 GB
Total #layers	5672
Total size of unique layers	74.25 GB



(a) CDF of the number of layers per image

(b) CDF of layer size (byte)

Fig. 1. The characteristics of layers.

C. Node storage capacity

For each network, we limit the nodes' capacities according to the total dataset size and number of nodes. First, the theoretical minimum node capacity that is needed to store all the layers (considering that the layers can be split at a byte level) is equal to the dataset size (with replication) divided by the number of nodes. However, this capacity will not satisfy any placement in practice as the integrity of the layers should be preserved. This can be achieved by storing only a complete layer on the same node. Therefore, we set the nodes' capacities as the theoretical minimum capacity multiplied by a *capacity scaling* factor. In our experiments, we test the following values for the *capacity scaling* factor: 1.1, 2, and INF. We include INF – which represents unlimited storage – just for comparison.

D. State-of-the-art placement algorithms

We compare our proposed algorithms with two placements algorithms: Random and Greedy (Best-Fit). These algorithms are not network-aware (i.e., they do not take into account links bandwidths). They serve as a comparison baseline.

1) Best-Fit placement: The Best-Fit placement is a greedy algorithm to place layers on nodes. The algorithm places the replicas of a layer l_i on the n_i nodes with the largest remaining storage capacity. The algorithm iterates over all the layers sorted by their decreasing size. As a result, Best-Fit distributes the layers evenly on the nodes in such a way that the nodes have almost the same total storage cost. When the layers have the same size, the behavior of Best-Fit will be like that of a round robin distribution. The algorithm is deterministic when the nodes initially presented in the same order, therefore, we shuffle the initial nodes ordering in every iteration to get a different placement.

2) Random placement: The Random placement serves as a very base solution. The algorithm distributes the layers randomly on the nodes. For each layer, we filter out the nodes that do not have sufficient storage space to host the layers, and then we select r random nodes to place the layer's replicas.

E. Methodology

For our experiments, we consider a default replication factor of 3 (as many storage systems [32]) for each layer ($n_i = 3$ for all i), therefore, the total dataset size with replication is $3 \times 74.25GB$. For KCBP-WC, we set the limit to define the "large" layers to 10%. For Best-Fit and Random placement algorithms, we run the placement 50 times and we draw the average retrieval times and the variation. KCBP and KCBP-WC are deterministic.

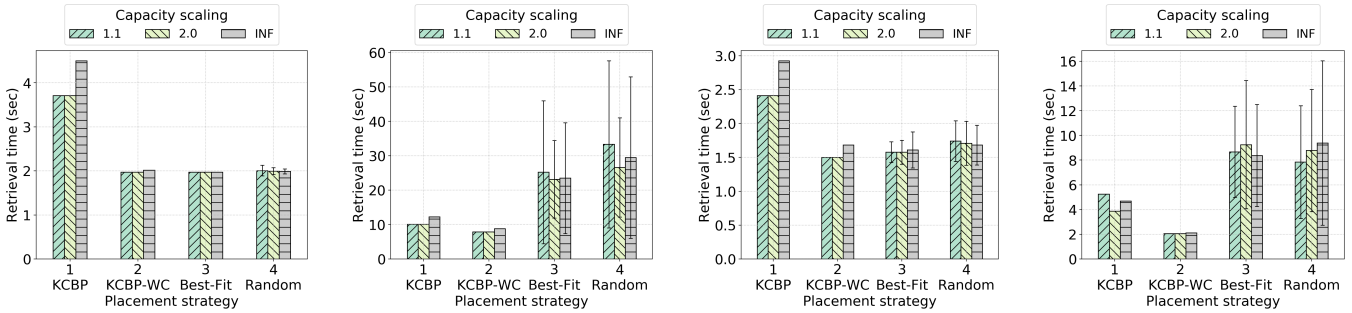
VI. SIMULATION RESULTS

In this section, we present the results of our simulations on the container images dataset presented earlier. We first focus on synthetic networks before considering real-world networks.

A. Results for Synthetic Networks

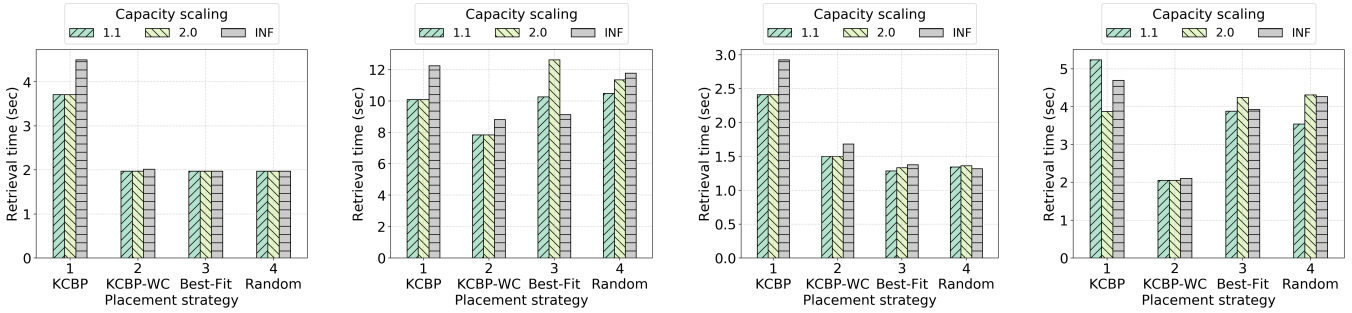
We would like to note that even though the retrieval times are presented in seconds, their relative values are more important than their absolute ones as the absolute retrieval time depends on the networks bandwidths. Similarly, for the synthetic networks, the distribution of link bandwidths is the important factor, not their actual values.

1) Homogeneous Network: All the links in the Homogeneous network have the same bandwidth, therefore, all the nodes have the same priority to place a layer in case of network connectivity. Fig. 2a shows the retrieval times of the placement algorithms when varying the capacity scaling factor. When the capacity scaling factor is set to 2, KCBP-WC has a maximum retrieval time of 1.97s, which is 1.8x faster than KCBP that needs 3.71s. With homogeneous link bandwidths, placing layers of the same image on the same node can prolong the retrieval time as in the case of KCBP. KCBP-WC handles this by distributing the layers of the same images. Moreover, as the bandwidths are homogeneous, we can notice that the performance of KCBP-WC is similar to Best-Fit and Random because the links bandwidths have no impact on the optimal



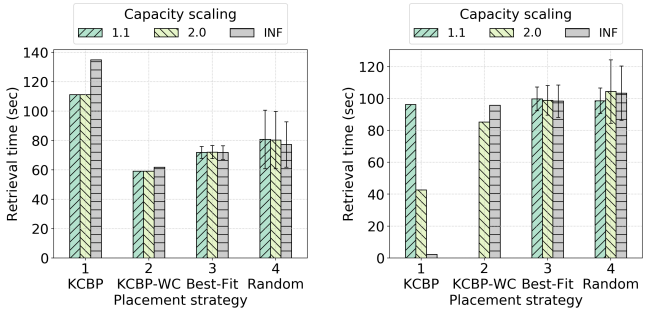
(a) Homogeneous Network (b) Low Network (c) High Network (d) Uniform Network

Fig. 2. Retrieval time for Synthetic networks.



(a) Homogeneous Network (b) Low Network (c) High Network (d) Uniform Network

Fig. 3. Retrieval time for Synthetic networks where the best obtained solution is shown for Best-Fit and Random.



(a) Renater Network (b) Sanet Network

Fig. 4. Retrieval time for Real-world Networks.

placement. Best-Fit has the same maximum retrieval time of KCBP-WC (i.e., 1.97*s*), while the maximum retrieval time of Random is 2.05*s*.

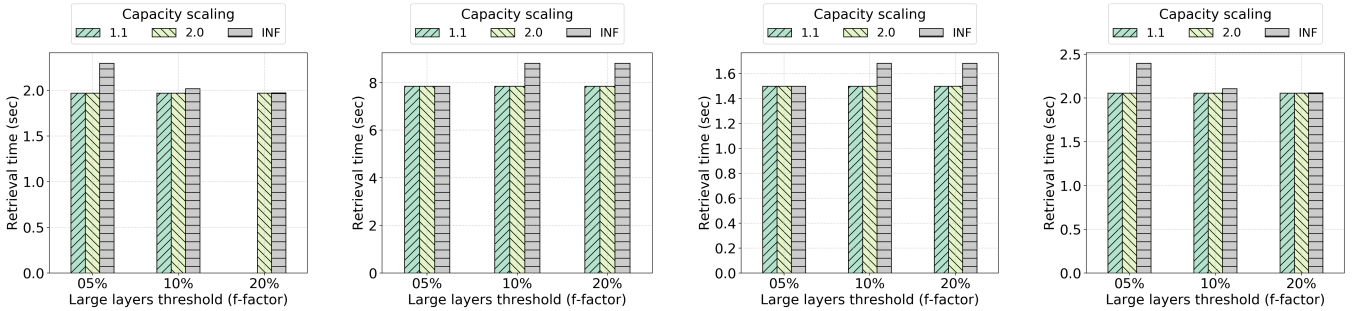
2) Low Network: In this network, the majority of the nodes are not well connected, therefore, the placement of the layers (especially large ones) is critical for the retrieval performance. Best-Fit and Random experience a high variation in retrieval times and their best-found placements are still worse than that of KCBP-WC (Fig. 2b). For example, for a capacity scaling factor of 2, KCBP-WC achieves 7.85*s* while KCBP requires 10.09*s*. Best-Fit and Random have an average retrieval time of 23.15*s* and 26.63*s*, while their best retrieval times are 12.63*s* and 11.35*s*, respectively.

3) High Network: With High network, the majority of links have high bandwidths, which shows that there are many nodes that are well connected to the rest. In contrary to the Low network, the probability of placing a "large" layer on a low-connected node is smaller, and therefore, we notice a smaller variation in the performance for Best-Fit and Random. Moreover, Best-Fit and Random have better retrieval times than KCBP-WC in their best case. They achieve 1.34*s* and 1.36*s*, respectively, while KCBP-WC has a retrieval time of 1.50*s* in case of capacity scaling factor of 2. Fig. 2c depicts the results.

4) Uniform Network: The Uniform Network (Fig. 2d) shows a similar trend to the Low network as both networks have a high percentage of low-bandwidth links and therefore low-connected nodes. We can notice that Best-Fit and Random exhibit high variation and KCBP-WC has better retrieval time even compared to their best case.

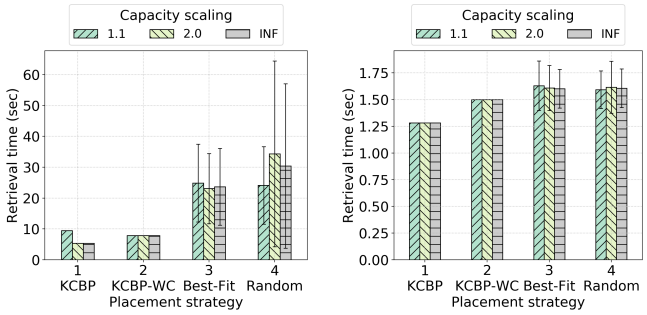
B. Results for Real-world Networks

1) Renater Network: Renater Network exhibits only small variations for links bandwidths, therefore, it shows similar behavior to Homogeneous network. For example, as we can see in Fig. 4a, for a capacity scaling factor of 2 KCBP-WC has a maximum retrieval time of 59*s* while KCBP achieve 111*s*, that is more or less the ratio expected according to previous results on homogeneous bandwidth. However, even if the bandwidths are more or less homogeneous, the existing small variations result in differences between the performance of KCBP-WC and the ones of Best-Fit and Best-Fit has a



(a) Homogeneous Network (b) Low Network (c) High Network (d) Uniform Network

Fig. 5. Retrieval time for KCBP-WC algorithm with different values for the f parameter on Synthetic networks.



(a) Low Network (b) High Network

Fig. 6. Retrieval time of individual Layers for Synthetic networks.

maximum retrieval time of 72s, which is 22% worse than KCBP-WC.

2) Sanet Network: With Sanet network, the majority of links have high bandwidth (the bandwidths of 75% of the links are higher than 6 Gbps). Thus the results are close to that of High network. However, contrary to the High network, in this setup, KCBP performs better with increasing nodes capacities. For example, it achieves 96s, 42s, and 2.3s for 1.1, 2, and INF capacity scaling factor, respectively (Fig. 4b). The main difference between Sanet and High network is the number of nodes (i.e., 35 against 50). Thus, the chance of placing some layers on low-connected nodes with KCBP-WC increases as does the retrieval time (85s for capacity scaling factor of 2). In such a case, pulling more than one layer from the same node is better than having a layer on a low-connected node.

VII. Discussion

In this section, we discuss the previous results and highlight our findings. We focus on five aspects: conflicts, heterogeneity of the bandwidth, storage capacity, percentage of layers considered as large for KCBP-WC, and maximal retrieval time per image.

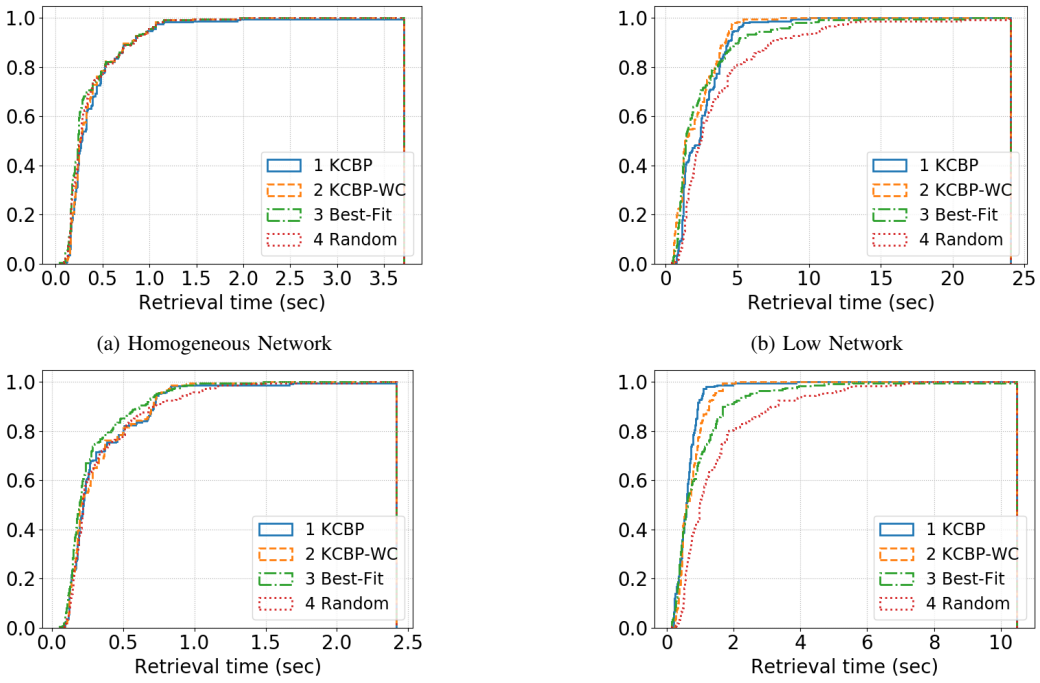
A. Impact of conflicts

In Fig. 6, we provide the maximum retrieval time for layers instead of images in order to evaluate the impact of conflict. As expected all strategies are not impacted the same way. For example, KCBP doubles its retrieval time when images are considered instead of individual layers on the High network (Fig. 2c and Fig. 6b). However, for KCBP-WC, the impact is small (more obvious with INF capacity) as the algorithm avoids as much as possible putting layers of the same image on the same node. In the case of Random and Best-Fit, as they tend to produce close to even distribution of layers on nodes, the probability of having more than one "large" layer on the same node is less than the case of KCBP and KCBP-WC, thus, conflicts between large layers are rare.

Thus, in general, avoiding conflict is an important factor to consider while placing replicas.

B. Impact of the heterogeneity of the bandwidth

As expected, having nodes with different connectivity change the behavior of the placement strategies. Here, bandwidth-aware strategies are able to deal more efficiently with this heterogeneity, even when it is small (as in case of Renater network). However, we note some important differences between High and Low networks. In the first case, the difference in retrieval times between KCBP-WC and average values of Best-Fit or Random is small, and the variation in the performances of the last two is rather low. For Low network, this variation greatly increases as does the average performance. In this case, the performance of KCBP is even close to KCBP-WC. However, this does not apply to High network. Hence, it seems that centrality of layers is more important for Low network than for High network. In the first case, it is important to target a few nodes with high connectivity while in the second it is important to avoid the few nodes with low connectivity (at least for the largest layers). Thus, in Low network, KCBP compensates the conflicts with a general good connectivity in comparison to the other node, while in High network, the node used by KCBP are not that much better than the average ones. At the same time, it is easier for Best-Fit and Random to avoid few low connectivity nodes than to reach the few high connectivity ones. KCBP-WC is able to manage both situations when the number of nodes is sufficient, otherwise, as in Sanet, it may suffer from the spreading of the layers, as Best-Fit and Random do. Note that in Uniform network, where bandwidths differ significantly, KCBP-WC and KCBP perform well, even against the best results from Best-Fit and Random for KCBP-WC.



(a) Homogeneous Network

(b) Low Network

(c) High Network

(d) Uniform Network

Fig. 7. CDF of maximal retrieval time per image with storage capacity scaling factor of 2.

C. The extra space effect

A phenomenon that is interesting to point out is the fact that the performances of KCBP and KCBP-WC can be improved by decreasing the node capacity. For example, this effect is visible for KCBP-WC on the Homogeneous network, and for KCBP on all the networks. The reason for this phenomenon is that several layers of the same image are more likely to be placed on the same node (and thus retrieved from the same node) when the node capacity is larger. As explained earlier, we proposed KCBP-WC to avoid such placements that are common with KCBP. However, in order to avoid having layers dispersed on too many nodes (or not being able to place all layers), we only apply this strategy on "large" layers, implying that, in some cases, the placement of layers of the same image still happens and thus slightly decreases the performances.

D. Impact of the percentage of layers concerned by KCBP-WC mechanism

Increasing the percentage of layers considered as "large" in KCBP-WC leads to a diminution of conflicts that should result in decreasing the retrieval time. However, it may also spread layers on nodes with low connectivity, leading to potentially longer retrieval times for some layers (and thus images). The extreme case is when there are no remaining nodes with enough storage capacity. As a result, KCBP-WC does not return a valid placement. This was the case on Sanet network which has only a small number of nodes (Fig. 4b). When testing different values for this percentage (5%, 10%, and 20%) as shown in Fig. 5, we observe almost no difference between them, except that with 20%. Moreover, the algorithm does not succeed to find a solution when the capacity factor is 1.1 on Homogeneous network.

We can not give a general conclusion from this result, as it is strongly correlated with the container images dataset, but it appears here that avoiding conflicts between only the largest layers is enough and expanding this policy to smaller layer offer no clear gain.

E. Maximal retrieval time per image

In this subsection, we discuss the maximal retrieval time per image (i.e., the maximal time to retrieve an image to any node). The cumulative distribution functions (CDFs) are presented in Fig. 7 for synthetic networks and capacity scaling factor of 2. We notice that the performances are close for all strategies in case of Homogeneous and High networks (Fig. 7a and Fig. 7c), with Best-Fit performing slightly better for a portion of images (i.e., around 20% of the images on High network). On such networks, the overall good quality of links does not favor strategies that aim to spread as little as possible the different replicas, as pointed out earlier when evaluating the impact of bandwidth heterogeneity. In addition, KCBP and KCBP-WC favor the largest layers, that may slightly degrade performances for images with smaller layers. Combined together, these two reasons fully explain these results in comparison to the overall equal distribution of layers among the nodes proposed by Best-Fit (even if the difference is not that important). However, for Low and Uniform networks (Fig. 7b and Fig. 7d) the trends are different. In these networks, centrality is important and thus KCBP and KCBP-WC perform well, even when we are considering other images than the ones with maximal retrieval times. More precisely, KCBP and

KCBP-WC present better maximal retrieval times for 20% of the images (images with longest retrieval time), with a small advantage for KCBP-WC that also performs better for other images. On Uniform network, KCBP outperforms all other strategies, at the exception of KCBP-WC that has finally a better overall maximum retrieval time (Fig. 2d).

From these distribution functions, we observe that although KCBP and KCBP-WC mainly target large layers (and the images they belong to), their performances are "good enough" compared to Best-Fit, when considering all images. Note that, Best-Fit can propose better retrieval times for intermediate images when network bandwidth is overall high. In a network with lower connectivity, the centralization of layers we propose with KCBP and KCBP-WC allows general improvement of maximal retrieval time for images.

VIII. Conclusion

Images management in Edge, especially container images, is gaining more importance with the widespread of Edge-servers. In this work, we propose to store the images across these Edge-servers, in a way that the missing layers of an image could be retrieved from nearby Edge-servers. The main goal behind this approach is to ensure predictable and reduced service provisioning time. To this end, we have proposed two image placement algorithms based on k-Center optimization to reduce the maximum retrieval time for an image to any Edge-server. Through extensive simulation, using synthetic and realistic networks with production container images dataset, we have shown that our proposed algorithm can reduce the maximum provisioning time by 1.1x to 4x compared to Random and Best-Fit based placements.

In considering future work, we plan to implement our solution in Docker and test its applicability on a real testbed. In addition, we will investigate how to address path sharing and the load variation of the network when retrieving a single and multiple container images.

Acknowledgment

This work is supported by the Inria Project Lab program Discovery (see http://beyondtheclouds.github.io), the Stack/Apollo connect talent project, and the ANR KerStream project (ANR-16-CE25-0014-01).

References

[1] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, 2015.

[2] E. G. Renart, J. Diaz-Montes, and M. Parashar, "Data-Driven Stream Processing at the Edge," in *ICFEC*, 2017.

[3] A. Ahmed and G. Pierre, "Docker Container Deployment in Fog Computing Infrastructures," in *EDGE*, 2018.

[4] (2017) Akamai's state of the internet report. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf

[5] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog Computing: Platform and Applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, 2015.

[6] (2018, Feb.) Real-time Video Analytics: the killer app for edge computing. [Online]. Available: https://www.microsoft.com/en-us/research/publication/real-time-video-analytics-killer-app-edge-computing/

[7] A. Anwar, M. Mohamed, V. Tarasov, M. Littley, L. Rupprecht, Y. Cheng, N. Zhao, D. Skourtis, A. S. Warke, H. Ludwig, D. Hildebrand, and A. R. Butt, "Improving Docker Registry Design Based on Production Workload Analysis," in *FAST*, 2018.

[8] (2019) The internet topology zoo. [Online]. Available: www.topology-zoo.org

[9] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[10] V. Bahl. (2015, May) Emergence of micro datacenter (cloudlets/edges) for mobile computing. [Online]. Available: https://www.microsoft.com/en-us/research/publication/emergence-of-micro-datacenter-cloudlets-edges-for-mobile-computing/

[11] N. Mohamed, J. Al-Jaroodi, I. Jawhar, S. Lazarova-Molnar, and S. Mahmoud, "SmartCityWare: A Service-Oriented Middleware for Cloud and Fog Enabled Smart City Services," *IEEE Access*, vol. 5, 2017.

[12] B. I. Ismail, E. M. Goortani, M. B. A. Karim, W. M. Tat, S. Setapa, J. Y. Luke, and O. H. Hoe, "Evaluation of Docker as Edge computing platform," in *ICOS*, 2015.

[13] P. Liu, D. Willis, and S. Banerjee, "ParaDrop: Enabling Lightweight Multi-tenancy at the Network's Extreme Edge," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2016, pp. 1–13.

[14] (2018, Feb.) Linuxcontainers.org, infrastructure for container projects. [Online]. Available: https://linuxcontainers.org

[15] (2018, Feb.) Docker homepage. [Online]. Available: https://www.docker.com

[16] (2019) Docker hub. [Online]. Available: https://hub.docker.com

[17] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy docker containers," in *FAST*, 2016.

[18] W. Kangjin, Y. Yong, L. Ying, L. Hanmei, and M. Lin, "FID: A Faster Image Distribution System for Docker Platform," in *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2017.

[19] S. Nathan, R. Ghosh, T. Mukherjee, and K. Narayanan, "CoMICon: A Co-Operative Management System for Docker Container Images," in *IC2E*, 2017.

[20] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *SYSTOR*, 2009.

[21] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei, "An Empirical Analysis of Similarity in Virtual Machine Images," in *Middleware*, 2011.

[22] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh, and D. Rubenstein, "VMTorrent: Scalable P2P Virtual Machine Streaming," in *CoNEXT*, 2012.

[23] S. Bazarbayev, M. Hiltunen, K. Joshi, W. H. Sanders, and R. Schlichting, "Content-Based Scheduling of Virtual Machines (VMs) in the Cloud," in *ICDCS*, 2013.

[24] X. Xu, H. Jin, S. Wu, and Y. Wang, "Rethink the storage of virtual machine images in clouds," *Future Gener. Comput. Syst.*, 2015.

[25] A. Karve and A. Kochut, "Redundancy Aware Virtual Disk Mobility for Cloud Computing," in *IEEE CLOUD*, 2013.

[26] J. Darrous, S. Ibrahim, A. Zhou, and C. Perez, "Nitro: Network-Aware Virtual Machine Image Management in Geo-Distributed Clouds," in *CCGrid*, 2018.

[27] M. R. Garey and D. S. Johnson, *Computers and intractability*. wh freeman New York, 2002, vol. 29.

[28] W.-L. Hsu and G. L. Nemhauser, "Easy and hard bottleneck location problems," *Discrete Applied Mathematics*, vol. 1, no. 3, pp. 209–215, 1979.

[29] B. Robič and J. Mihelič, "Solving the k-center problem efficiently with a dominating set algorithm," *Journal of computing and information technology*, vol. 13, no. 3, pp. 225–234, 2005.

[30] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-driven WAN," in *SIGCOMM*, 2013.

[31] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases and Future Directions," *IEEE Communications Surveys and Tutorials*, vol. PP, pp. 1–1, 06 2017.

[32] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *MSST*, 2010.